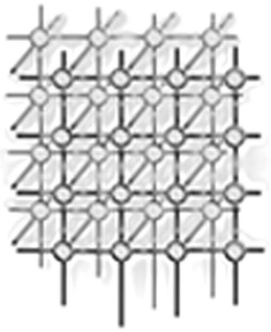
Reclaiming the energy of a schedule: models and algorithms

Guillaume Aupy, Anne Benoit, Fanny Dufossé, Yves Robert

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2011; **00**:1–20 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*



Reclaiming the energy of a schedule: models and algorithms

Guillaume Aupy[1], Anne Benoit*[1,2], Fanny Dufossé[1] and Yves Robert[1,2]

[1] *ENS Lyon, Université de Lyon, LIP laboratory, UMR 5668, ENS Lyon–CNRS–INRIA–UCBL, Lyon, France*
[2] *Institut Universitaire de France*

SUMMARY

We consider a task graph to be executed on a set of processors. We assume that the mapping is given, say by an ordered list of tasks to execute on each processor, and we aim at optimizing the energy consumption while enforcing a prescribed bound on the execution time. While it is not possible to change the allocation of a task, it is possible to change its speed. Rather than using a local approach such as backfilling, we consider the problem as a whole and study the impact of several speed variation models on its complexity. For continuous speeds, we give a closed-form formula for trees and series-parallel graphs, and we cast the problem into a geometric programming problem for general directed acyclic graphs (DAGs). We show that the classical dynamic voltage and frequency scaling (DVFS) model with discrete modes leads to a NP-complete problem, even if the modes are regularly distributed (an important particular case in practice, which we analyze as the incremental model). On the contrary, the Vdd-hopping model that allows to switch between different supply voltages (V_{DD}) while executing a task leads to a polynomial solution. Finally, we provide an approximation algorithm for the incremental model, which we extend for the general DVFS model.

KEY WORDS: Energy models, complexity, bi-criteria optimization, algorithms, scheduling.

*Correspondence to: Anne Benoit, LIP, ENS Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France.
†E-mail: {Guillaume.Aupy, Anne.Benoit, Fanny.Dufosse, Yves.Robert}@ens-lyon.fr.
This work was supported in part by the ANR StochaGrid and RESCUE projects. A two-page extended abstract of this work appears as a short presentation in SPAA'2011.

1. Introduction

The *energy consumption* of computational platforms has recently become a critical problem, both for economic and environmental reasons [1]. As an example, the Earth Simulator requires about 12 MW (Mega Watts) of peak power, and PetaFlop systems may require 100 MW of power, nearly the output of a small power plant (300 MW). At $100 per MW.Hour, peak operation of a PetaFlop machine may thus cost $10,000 per hour [2]. Current estimates state that cooling costs $1 to $3 per watt of heat dissipated [3]. This is just one of the many economical reasons why energy-aware scheduling has proved to be an important issue in the past decade, even without considering battery-powered systems such as laptops and embedded systems. As an example, the Green500 list (`www.green500.org`) provides rankings of the most energy-efficient supercomputers in the world, therefore raising even more awareness about power consumption.

To help reduce energy dissipation, processors can run at different speeds. Their power consumption is the sum of a static part (the cost for a processor to be turned on) and a dynamic part, which is a strictly convex function of the processor speed, so that the execution of a given amount of work costs more power if a processor runs in a higher mode [4]. More precisely, a processor running at speed s dissipates s^3 watts [5, 6, 7, 8, 9] per time-unit, hence consumes $s^3 \times d$ joules when operated during d units of time. Faster speeds allow for a faster execution, but they also lead to a much higher (supra-linear) power consumption.

Energy-aware scheduling aims at minimizing the energy consumed during the execution of the target application. Obviously, it makes sense only if it is coupled with some performance bound to achieve, otherwise, the optimal solution always is to run each processor at the slowest possible speed.

In this paper, we investigate energy-aware scheduling strategies for executing a task graph on a set of processors. The main originality is that we assume that the mapping of the task graph is given, say by an ordered list of tasks to execute on each processor. There are many situations in which this problem is important, such as optimizing for legacy applications, or accounting for affinities between tasks and resources, or even when tasks are pre-allocated [10], for example for security reasons. In such situations, assume that a list-schedule has been computed for the task graph, and that its execution time should not exceed a deadline D. We do not have the freedom to change the assignment of a given task, but we can change its speed to reduce energy consumption, provided that the deadline D is not exceeded after the speed change. Rather than using a local approach such as backfilling [11, 12], which only reclaims gaps in the schedule, we consider the problem as a whole, and we assess the impact of several speed variation models on its complexity. More precisely, we investigate the following models:

Continuous model. Processors can have arbitrary speeds, and can vary them continuously: this model is unrealistic (any possible value of the speed, say $\sqrt{e^\pi}$, cannot be obtained) but it is theoretically appealing [13]. A maximum speed, s_{max}, cannot be exceeded.

Discrete model. Processors have a discrete number of predefined speeds (or frequencies), which correspond to different voltages that the processor can be subjected to [14]. Switching frequencies is not allowed during the execution of a given task, but two different tasks scheduled on a same processor can be executed at different frequencies.

Vdd-Hopping model. This model is similar to the DISCRETE one, except that switching modes during the execution of a given task is allowed: any rational speed can be simulated, by simply switching, at the appropriate time during the execution of a task, between two consecutive modes [15]. Note that V_{DD} usually represents the supply voltage, hence the name VDD-HOPPING.

Incremental model. In this variant of the DISCRETE model, we introduce a value δ that corresponds the minimum permissible speed increment, induced by the minimum voltage increment that can be achieved when controlling the processor CPU. This new model aims at capturing a realistic version of the DISCRETE model, where the different modes are spread regularly instead of arbitrarily chosen.

Our main contributions are the following. For the CONTINUOUS model, we give a closed-form formula for trees and series-parallel graphs, and we cast the problem into a geometric programming problem [16] for general DAGs. For the VDD-HOPPING model, we show that the optimal solution for general DAGs can be computed in polynomial time, using a (rational) linear program. Finally, for the DISCRETE and INCREMENTAL models, we show that the problem is NP-complete. Furthermore, we provide approximation algorithms that rely on the polynomial algorithm for the VDD-HOPPING model, and we compare their solution with the optimal CONTINUOUS solution.

The paper is organized as follows. We start with a survey of related literature in Section 2. We then provide the formal description of the framework and of the energy models in Section 3, together with a simple example to illustrate the different models. The next two sections constitute the heart of the paper: in Section 4, we provide analytical formulas for continuous speeds, and the formulation into the convex optimization problem. In Section 5, we assess the complexity of the problem with all the discrete models: DISCRETE, VDD-HOPPING and INCREMENTAL, and we discuss approximation algorithms. Finally we conclude in Section 6.

2. Related work

Reducing the energy consumption of computational platforms is an important research topic, and many techniques at the process, circuit design, and micro-architectural levels have been proposed [17, 18, 19]. The dynamic voltage and frequency scaling (DVFS) technique has been extensively studied, since it may lead to efficient energy/performance trade-offs [20, 2, 13, 21, 22, 23, 11]. Current microprocessors (for instance, from AMD [24] and Intel [25]) allow the speed to be set dynamically. Indeed, by lowering supply voltage, hence processor clock frequency, it is possible to achieve important reductions in power consumption, without necessarily increasing the execution time. We first discuss different optimization problems that arise in this context. Then we review energy models.

2.1. DVFS and optimization problems

When dealing with energy consumption, the most usual optimization function consists in minimizing the energy consumption, while ensuring a deadline on the execution time (i.e., a real-time constraint), as discussed in the following papers.

In [14], Okuma et al. demonstrate that voltage scaling is far more effective than the shutdown approach, which simply stops the power supply when the system is inactive. Their target processor employs just a few discretely variable voltages. De Langen and Juurlink [26] discuss leakage-aware scheduling heuristics that investigate both dynamic voltage scaling (DVS) and processor shutdown, since static power consumption due to leakage current is expected to increase significantly. Chen et al. [27] consider parallel sparse applications, and they show that when scheduling applications modeled by a directed acyclic graph with a well-identified critical path, it is possible to lower the voltage during non-critical execution of tasks, with no impact on the execution time. Similarly, Wang et al. [11] study the slack time for non-critical jobs, they extend their execution time and thus reduce the energy consumption without increasing the total execution time. Kim et al. [22] provide power-aware scheduling algorithms for bag-of-tasks applications with deadline constraints, based on dynamic voltage scaling. Their goal is to minimize power consumption as well as to meet the deadlines specified by application users.

For real-time embedded systems, slack reclamation techniques are used. Lee and Sakurai [17] show how to exploit slack time arising from workload variation, thanks to a software feedback control of supply voltage. Prathipati [12] discusses techniques to take advantage of run-time variations in the execution time of tasks; it determines the minimum voltage under which each task can be executed, while guaranteeing the deadlines of each task. Then, experiments are conducted on the Intel StrongArm SA-1100 processor, which has eleven different frequencies, and the Intel PXA250 XScale embedded processor with four frequencies. In [28], the goal of Xu et al. is to schedule a set of independent tasks, given a worst case execution cycle (WCEC) for each task, and a global deadline, while accounting for time and energy penalties when the processor frequency is changing. The frequency of the processor can be lowered when some slack is obtained dynamically, typically when a task runs faster than its WCEC. Yang and Lin [23] discuss algorithms with preemption, using DVS techniques; substantial energy can be saved using these algorithms, which succeed to claim the static and dynamic slack time, with little overhead.

Since an increasing number of systems are powered by batteries, maximizing battery life also is an important optimization problem. Battery-efficient systems can be obtained with similar techniques of dynamic voltage and frequency scaling, as described by Lahiri et al. in [18]. Another optimization criterion is the energy-delay product, since it accounts for a trade-off between performance and energy consumption, as for instance discussed by Gonzalez and Horowitz in [29]. We do not discuss further these latter optimization problems, since our goal is to minimize the energy consumption, with a fixed deadline.

In this paper, the application is a task graph (directed acyclic graph), and we assume that the mapping, i.e., an ordered list of tasks to execute on each processor, is given. Hence, our problem is closely related to slack reclamation techniques, but instead on focusing on non-critical tasks as for instance in [11], we consider the problem as a whole. Our contribution is

to perform an exhaustive complexity study for different energy models. In the next paragraph, we discuss related work on each energy model.

2.2. Energy models

Several energy models are considered in the literature, and they can all be categorized in one of the four models investigated in this paper, i.e., CONTINUOUS, DISCRETE, VDD-HOPPING or INCREMENTAL.

The CONTINUOUS model is used mainly for theoretical studies. For instance, Yao et al. [30], followed by Bansal et al. [13], aim at scheduling a collection of tasks (with release time, deadline and amount of work), and the solution is the time at which each task is scheduled, but also, the speed at which the task is executed. In these papers, the speed can take any value, hence following the CONTINUOUS model.

We believe that the most widely used model is the DISCRETE one. Indeed, processors have currently only a few discrete number of possible frequencies [24, 25, 14, 12]. Therefore, most of the papers discussed above follow this model. Some studies exploit the continuous model to determine the smallest frequency required to run a task, and then choose the closest upper discrete value, as for instance [12] and [31].

Recently, a new local dynamic voltage scaling architecture has been developed, based on the VDD-HOPPING model [15, 32, 33]. It was shown in [17] that significant power can be saved by using two distinct voltages, and architectures using this principle have been developed (see for instance [34]). Compared to traditional power converters, a new design with no needs for large passives or costly technological options has been validated in a STMicroelectronics CMOS 65nm low-power technology [15].

To the best of our knowledge, this paper introduces the INCREMENTAL model for the first time. The main rationale is that future technologies may well have an increased number of possible frequencies, and these will follow a regular pattern. For instance, note that the SA-1100 processor, considered in [12], has eleven frequencies that are equidistant, i.e., they follow the INCREMENTAL model. Lee and Sakurai [17] exploit discrete levels of clock frequency as f, $f/2$, $f/3$, ..., where f is the master (i.e., the higher) system clock frequency. This model is closer to the DISCRETE model, although it exhibits a regular pattern similarly to the INCREMENTAL model.

Our work is the first attempt to compare these different models: on the one hand, we assess the impact of the model on the problem complexity (polynomial vs NP-hard), and on the other hand, we provide approximation algorithms building upon these results. The closest work to ours is the paper by Zhang et al. [31], in which the authors also consider the mapping of directed acyclic graphs, and compare the DISCRETE and the CONTINUOUS models. We go beyond their work in this paper, with an exhaustive complexity study, closed-form formulas for the continuous model, and the comparison with the VDD-HOPPING and INCREMENTAL models.

3. Framework

First we detail the optimization problem in Section 3.1. Then we describe the four energy models in Section 3.2. Finally, we illustrate the models and motivate the problem with an example in Section 3.3.

3.1. Optimization problem

Consider an application task graph $\mathcal{G} = (V, \mathcal{E})$, with $n = |V|$ tasks denoted as $V = \{T_1, T_2, \ldots, T_n\}$, and where the set $\mathcal{E}$ denotes the precedence edges between tasks. Task T_i has a cost w_i for $1 \leq i \leq n$. We assume that the tasks in $\mathcal{G}$ have been allocated onto a parallel platform made up of identical processors. We define the *execution graph* generated by this allocation as the graph $G = (V, E)$, with the following augmented set of edges:

- $\mathcal{E} \subseteq E$: if an edge exists in the precedence graph, it also exists in the execution graph;
- if T_1 and T_2 are executed successively, in this order, on the same processor, then $(T_1, T_2) \in E$.

The goal is to the minimize the energy consumed during the execution while enforcing a deadline D on the execution time. We formalize the optimization problem in the simpler case where each task is executed at constant speed. This strategy is optimal for the CONTINUOUS model (by a convexity argument) and for the DISCRETE and INCREMENTAL models (by definition). For the VDD-HOPPING model, we reformulate the problem in Section 5.1. For each task $T_i \in V$, b_i is the starting time of its execution, d_i is the duration of its execution, and s_i is the speed at which it is executed. We obtain the following formulation of the MINENERGY(G, D) problem, given an execution graph $G = (V, E)$ and a deadline D; the s_i values are variables, whose values are constrained by the energy model (see Section 3.2).

$$\begin{array}{lll} \text{Minimize} & & \sum_{i=1}^{n} s_i^3 \times d_i \\ \text{subject to} & \text{(i)} & w_i = s_i \times d_i \text{ for each task } T_i \in V \\ & \text{(ii)} & b_i + d_i \leq b_j \text{ for each edge } (T_i, T_j) \in E \\ & \text{(iii)} & b_i + d_i \leq D \text{ for each task } T_i \in V \\ & \text{(iv)} & b_i \geq 0 \text{ for each task } T_i \in V \end{array} \tag{1}$$

Constraint (i) states that the whole task can be executed in time d_i using speed s_i. Constraint (ii) accounts for all dependencies, and constraint (iii) ensures that the execution time does not exceed the deadline D. Finally, constraint (iv) enforces that starting times are non-negative. The energy consumed throughout the execution is the objective function. It is the sum, for each task, of the energy consumed by this task, as we detail in the next section. Note that $d_i = w_i/s_i$, and therefore the objective function can also be expressed as $\sum_{i=1}^{n} s_i^2 \times w_i$.

Note that, whatever the energy model, there is a maximum speed that cannot be exceeded, denoted s_{max}. We point out that there is a solution to the minimization problem if and only if there is a solution with $s_i = s_{max}$ for all $1 \leq i \leq n$. Such a solution would correspond to executing each task as early as possible (according to constraints (ii) and (iv)) and as fast as possible. The optimal solution then slows down tasks to save as much energy as possible, while enforcing the deadline constraint. There is no guarantee on the uniqueness of the solution,

since it may be possible to modify the beginning time of a task without affecting the energy consumption, if some of the constraints (ii) are not tight.

3.2. Energy models

In all models, when a processor operates at speed s during d time-units, the corresponding consumed energy is $s^3 \times d$, which is the dynamic part of the energy consumption, following the classical models of the literature [5, 6, 7, 8, 9]. Note that we do not take static energy into account, because all processors are up and alive during the whole execution. We now detail the possible speed values in each energy model, which should be added as a constraint in Equation (1).

- In the CONTINUOUS model, processors can have arbitrary speeds, from 0 to a maximum value s_{max}, and a processor can change its speed at any time during execution.
- In the DISCRETE model, processors have a set of possible speed values, or modes, denoted as $s_1, ..., s_m$. There is no assumption on the range and distribution of these modes. The speed of a processor cannot change during the computation of a task, but it can change from task to task.
- In the VDD-HOPPING model, a processor can run at different speeds $s_1, ..., s_m$, as in the previous model, but it can also change its speed during a computation. The energy consumed during the execution of one task is the sum, on each time interval with constant speed s, of the energy consumed during this interval at speed s.
- In the INCREMENTAL model, we introduce a value δ that corresponds to the minimum permissible speed (i.e., voltage) increment. That means that possible speed values are obtained as $s = s_{min} + i \times \delta$, where i is an integer such that $0 \leq i \leq \frac{s_{max} - s_{min}}{\delta}$. Admissible speeds lie in the interval $[s_{min}, s_{max}]$. This new model aims at capturing a realistic version of the DISCRETE model, where the different modes are spread regularly between $s_1 = s_{min}$ and $s_m = s_{max}$, instead of being arbitrarily chosen. It is intended as the modern counterpart of a potentiometer knob!

3.3. Example

Consider an application with four tasks of costs $w_1 = 3$, $w_2 = 2$, $w_3 = 1$ and $w_4 = 2$, and one precedence constraint $T_1 \rightarrow T_3$. We assume that T_1 and T_2 are allocated, in this order, onto processor P_1, while T_3 and T_4 are allocated, in this order, on processor P_2. The resulting execution graph G is given in Figure 1, with two precedence constraints added to the initial task graph. The deadline on the execution time is $D = 1.5$.

We set the maximum speed to $s_{max} = 6$ for the CONTINUOUS model. For the DISCRETE and VDD-HOPPING models, we use the set of speeds $s_1^{(d)} = 2$, $s_2^{(d)} = 5$ and $s_3^{(d)} = 6$. Finally, for the INCREMENTAL model, we set $\delta = 2$, $s_{min} = 2$ and $s_{max} = 6$, so that possible speeds are $s_1^{(i)} = 2$, $s_2^{(i)} = 4$ and $s_3^{(i)} = 6$. We aim at finding the optimal execution speed s_i for each task T_i ($1 \leq i \leq 4$), i.e., the values of s_i that minimize the energy consumption.

With the CONTINUOUS model, the optimal speeds are non rational values, and we obtain

$$s_1 = \frac{2}{3}(3 + 35^{1/3}) \simeq 4.18; \quad s_2 = s_1 \times \frac{2}{35^{1/3}} \simeq 2.56; \quad s_3 = s_4 = s_1 \times \frac{3}{35^{1/3}} \simeq 3.83.$$

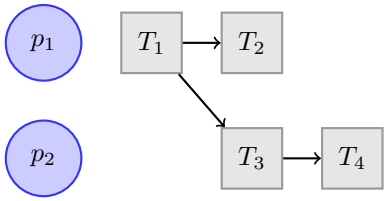


Figure 1: Execution graph for the example.

Note that all speeds are lower than the maximum s_{max}. These values are obtained thanks to the formulas derived in Section 4. The energy consumption is then $E_{opt}^{(c)} = \sum_{i=1}^{4} w_i \times s_i^2 = 3.s_1^2 + 2.s_2^2 + 3.s_3^2 \simeq 109.6$. The execution time is $\frac{w_1}{s_1} + \max\left(\frac{w_2}{s_2}, \frac{w_3+w_4}{s_3}\right)$, and with this solution, it is equal to the deadline D (actually, both processors reach the deadline, otherwise we could slow down the execution of one task).

For the DISCRETE model, if we execute all tasks at speed $s_2^{(d)} = 5$, we obtain an energy $E = 8 \times 5^2 = 200$. A better solution is obtained with $s_1 = s_3^{(d)} = 6$, $s_2 = s_3 = s_1^{(d)} = 2$ and $s_4 = s_2^{(d)} = 5$, which turns out to be optimal: $E_{opt}^{(d)} = 3 \times 36 + (2+1) \times 4 + 2 \times 25 = 170$. Note that $E_{opt}^{(d)} > E_{opt}^{(c)}$, i.e., the optimal energy consumption with the DISCRETE model is much higher than the one achieved with the CONTINUOUS model. Indeed, in this case, even though the first processor executes during $3/6 + 2/2 = D$ time units, the second processor remains idle since $3/6 + 1/2 + 2/5 = 1.4 < D$. The problem turns out to be NP-hard (see Section 5.2), and the solution has been found by performing an exhaustive search.

With the VDD-HOPPING model, we set $s_1 = s_2^{(d)} = 5$; for the other tasks, we run part of the time at speed $s_2^{(d)} = 5$, and part of the time at speed $s_1^{(d)} = 2$ in order to use the idle time and lower the energy consumption. T_2 is executed at speed $s_1^{(d)}$ during time $\frac{5}{6}$ and at speed $s_2^{(d)}$ during time $\frac{2}{30}$ (i.e., the first processor executes during time $3/5 + 5/6 + 2/30 = 1.5 = D$, and all the work for T_2 is done: $2 \times 5/6 + 5 \times 2/30 = 2 = w_2$). T_3 is executed at speed $s_2^{(d)}$ (during time $1/5$), and finally T_4 is executed at speed $s_1^{(d)}$ during time 0.5 and at speed $s_2^{(d)}$ during time $1/5$ (i.e., the second processor executes during time $3/5 + 1/5 + 0.5 + 1/5 = 1.5 = D$, and all the work for T_4 is done: $2 \times 0.5 + 5 \times 1/5 = 2 = w_4$). This set of speeds turns out to be optimal (i.e., it is the optimal solution of the linear program introduced in Section 5.1), with an energy consumption $E_{opt}^{(v)} = (3/5 + 2/30 + 1/5 + 1/5) \times 5^3 + (5/6 + 0.5) \times 2^3 = 144$. As expected, $E_{opt}^{(c)} \leq E_{opt}^{(v)} \leq E_{opt}^{(d)}$, i.e., the VDD-HOPPING solution stands between the optimal CONTINUOUS solution, and the more constrained DISCRETE solution.

For the INCREMENTAL model, the reasoning is similar to the DISCRETE case, and the optimal solution is obtained by an exhaustive search: all tasks should be executed at speed $s_2^{(i)} = 4$, with an energy consumption $E_{opt}^{(i)} = 8 \times 4^2 = 128 > E_{opt}^{(c)}$. It turns out to be better than DISCRETE and VDD-HOPPING, since it has different discrete values of energy that are more appropriate for this example.

We conclude the study of this simple example with a short discussion on the energy savings that can be achieved. All three models have a maximum speed $s_{max} = 6$. Executing the four tasks at maximum speed leads to consuming an energy $E_{max} = 8 \times 6^2 = 288$. Such an execution completes within a delay $D = 1$. We clearly see the trade-off between execution time and energy consumption here, since we gain more than half the energy by slowing down the execution from $D = 1$ to $D = 1.5$. Note that with $D = 1$, we can still slow down task T_2 to speed 4, and still gain a little over the brute force solution. Hence, even such a toy example allows us to illustrate the benefits of energy-aware schedules. Obviously, with larger examples, the energy savings will be even more dramatic, depending upon the range of available speeds and the tightness of the execution deadline. In fact, the maximal energy gain that can be achieved is not bounded: when executing each task as slow as possible (instead of as fast as possible), we gain $\left(\frac{s_{max}}{s_{min}}\right)^2 W_{\text{total}}$, where W_{total} is the sum of all task weights, and this quantity can be arbitrarily large. One of the main contributions of this paper is to provide optimal energy-aware algorithms for each model (or guaranteed polynomial approximations for NP-complete instances).

4. The Continuous model

With the CONTINUOUS model, processor speeds can take any value between 0 and s_{max}. First we prove that, with this model, the processors do not change their speed during the execution of a task (Section 4.1). Then, we derive in Section 4.2 the optimal speed values for special execution graph structures, expressed as closed form algebraic formulas, and we show that these values may be irrational (as already illustrated in the example in Section 3.3). Finally, we formulate the problem for general DAGs as a convex optimization program in Section 4.3.

4.1. Preliminary lemma

Lemma 1 (constant speed per task) *In all optimal solution with the* CONTINUOUS *model, each task is executed at constant speed, i.e., a processor does not change its speed during the execution of a task.*

Proof. Suppose that in the optimal solution, there is a task whose speed changes during the execution. Consider the first time-step at which the change occurs: the computation begins at speed s from time t to time t', and then continues at speed s' until time t''. The total energy consumption for this task in the time interval $[t; t'']$ is $E = (t' - t) \times s^3 + (t'' - t') \times (s')^3$. Moreover, the amount of work done for this task is $W = (t' - t) \times s + (t'' - t') \times s'$.

If we run the task during the whole interval $[t; t'']$ at constant speed $W/(t'' - t)$, the same amount of work is done within the same time. However, the energy consumption during this interval of time is now $E' = (t'' - t) \times (W/(t'' - t))^3$. By convexity of the function $x \mapsto x^3$, we obtain $E' < E$ since $t < t' < t''$. This contradicts the hypothesis of optimality of the first solution, which concludes the proof. □

4.2. Special execution graphs

4.2.1. Independent tasks

Consider the problem of minimizing the energy of n independent tasks (i.e., each task is mapped onto a distinct processor, and there are no precedence constraints in the execution graph), while enforcing a deadline D.

Proposition 1 (independent tasks) *When G is composed of independent tasks $\{T_1, \ldots, T_n\}$, the optimal solution to* MinEnergy(G, D) *is obtained when each task T_i ($1 \leq i \leq n$) is computed at speed $s_i = \frac{w_i}{D}$. If there is a task T_i such that $s_i > s_{max}$, then the problem has no solution.*

Proof. For task T_i, the speed s_i corresponds to the slowest speed at which the processor can execute the task, so that the deadline is not exceeded. If $s_i > s_{max}$, the corresponding processor will never be able to complete its execution before the deadline, therefore there is no solution. To conclude the proof, we note that any other solution would meet the deadline constraint, and therefore the s_i's should be such that $\frac{w_i}{s_i} \leq D$, which means that $s_i \geq \frac{w_i}{D}$. These values would all be higher than the s_i's of the optimal solution, and hence would lead to a higher energy consumption. Therefore, this solution is optimal. □

4.2.2. Linear chain of tasks

This case corresponds for instance to n independent tasks $\{T_1, \ldots, T_n\}$ executed onto a single processor. The execution graph is then a linear chain (order of execution of the tasks), with $T_i \rightarrow T_{i+1}$, for $1 \leq i < n$.

Proposition 2 (linear chain) *When G is a linear chain of tasks, the optimal solution to* MinEnergy(G, D) *is obtained when each task is executed at speed $s = \frac{W}{D}$, with $W = \sum_{i=1}^{n} w_i$. If $s > s_{max}$, then there is no solution.*

Proof. Suppose that in the optimal solution, tasks T_i and T_j are such that $s_i < s_j$. The total energy consumption is E_{opt}. We define s such that the execution of both tasks running at speed s takes the same amount of time than in the optimal solution, i.e., $(w_i + w_j)/s = w_i/s_i + w_j/s_j$: $s = \frac{(w_i+w_j)}{w_i s_j + w_j s_i} \times s_i s_j$. Note that $s_i < s < s_j$ (it is the barycenter of two points with positive mass).

We consider a solution such that the speed of task T_k, for $1 \leq k \leq n$, with $k \neq i$ and $k \neq j$, is the same as in the optimal solution, and the speed of tasks T_i and T_j is s. By definition of s, the execution time has not been modified. The energy consumption of this solution is E, where $E_{opt} - E = w_i s_i^2 + w_j s_j^2 - (w_i + w_j)s^2$, i.e., the difference of energy with the optimal solution is only impacted by tasks T_i and T_j, for which the speed has been modified. By convexity of the function $x \mapsto x^2$, we obtain $E_{opt} > E$, which contradicts its optimality. Therefore, in the optimal solution, all tasks have the same execution speed. Moreover, the energy consumption is minimized when the speed is as low as possible, while the deadline is not exceeded. Therefore, the execution speed of all tasks is $s = W/D$. □

Corollary 1. *A linear chain with n tasks is equivalent to a single task of cost $W = \sum_{i=1}^{n} w_i$.*

Indeed, in the optimal solution, the n tasks are executed at the same speed, and they can be replaced by a single task of cost W, which is executed at the same speed and consumes the same amount of energy.

4.2.3. Fork and join graphs

Let $V = \{T_1, \ldots, T_n\}$. We consider either a fork graph $G = (V \cup \{T_0\}, E)$, with $E = \{(T_0, T_i), T_i \in V\}$, or a join graph $G = (V \cup \{T_0\}, E)$, with $E = \{(T_i, T_0), T_i \in V\}$. T_0 is either the source of the fork or the sink of the join.

Theorem 1 (fork and join graphs) *When G is a fork (resp. join) execution graph with $n+1$ tasks $T_0, T_1, \ldots, T_n$, the optimal solution to* MinEnergy(G, D) *is the following:*

- *the execution speed of the source (resp. sink) T_0 is $s_0 = \dfrac{\left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}} + w_0}{D}$;*
- *for the other tasks T_i, $1 \leq i \leq n$, we have $s_i = s_0 \times \dfrac{w_i}{\left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}}}$ if $s_0 \leq s_{max}$.*

Otherwise, T_0 should be executed at speed $s_0 = s_{max}$, and the other speeds are $s_i = \frac{w_i}{D'}$, with $D' = D - \frac{w_0}{s_{max}}$, if they do not exceed s_{max} (Proposition 1 for independent tasks). Otherwise there is no solution.

If no speed exceeds s_{max}, the corresponding energy consumption is

$$\mathbf{minE}(G, D) = \frac{\left(\left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}} + w_0\right)^3}{D^2} .$$

Proof. Let $t_0 = \frac{w_0}{s_0}$. Then, the source or the sink requires a time t_0 for execution. For $1 \leq i \leq n$, task T_i must be executed within a time $D - t_0$ so that the deadline is respected. Given t_0, we can compute the speed s_i for task T_i using Theorem 1, since the tasks are independent: $s_i = \frac{w_i}{D - t_0} = w_i \cdot \frac{s_0}{s_0 D - w_0}$. The objective is therefore to minimize $\sum_{i=0}^{n} w_i s_i^2$, which is a function of s_0:

$$\sum_{i=0}^{n} w_i s_i^2 = w_0 s_0^2 + \sum_{i=1}^{n} w_i^3 \cdot \frac{s_0^2}{(s_0 D - w_0)^2} = s_0^2 \left(w_0 + \frac{\sum_{i=1}^{n} w_i^3}{(s_0 D - w_0)^2}\right) = f(s_0).$$

Let $W_3 = \sum_{i=1}^{n} w_i^3$. In order to find the value of s_0 that minimizes this function, we study the function $f(x)$, for $x > 0$. $f'(x) = 2x\left(w_0 + \frac{W_3}{(xD - w_0)^2}\right) - 2D \cdot x^2 \cdot \frac{W_3}{(xD - w_0)^3}$, and therefore $f'(x) = 0$ for $x = (W_3^{\frac{1}{3}} + w_0)/D$. We conclude that the optimal speed for task T_0 is $s_0 = \frac{\left(\sum_{i=1}^{n} w_i^3\right)^{\frac{1}{3}} + w_0}{D}$, if $s_0 \leq s_{max}$. Otherwise, T_0 should be executed at the maximum speed $s_0 = s_{max}$, since it is the bottleneck task. In any case, for $1 \leq i \leq n$, the optimal speed for task T_i is $s_i = w_i \frac{s_0}{s_0 D - w_0}$.

Finally, we compute the exact expression of $\mathbf{minE}(G, D) = f(s_0)$, when $s_0 \leq s_{max}$:

$$f(s_0) = s_0^2 \left(w_0 + \frac{W_3}{(s_0 D - w_0)^2} \right) = \left(\frac{W_3^{\frac{1}{3}} + w_0}{D} \right)^2 \left(\frac{W_3}{W_3^{2/3}} + w_0 \right) = \frac{\left(W_3^{\frac{1}{3}} + w_0 \right)^3}{D^2},$$

which concludes the proof. □

Corollary 2 (equivalent tasks for speed) *Consider a fork or join graph with tasks T_i, $0 \leq i \leq n$, and a deadline D, and assume that the speeds in the optimal solution to* MinEnergy(G, D) *do not exceed s_{max}. Then, these speeds are the same as in the optimal solution for $n+1$ independent tasks $T'_0, T'_1, \ldots, T'_n$, where $w'_0 = \left(\sum_{i=1}^n w_i^3 \right)^{\frac{1}{3}} + w_0$, and, for $1 \leq i \leq n$, $w'_i = w'_0 \cdot \frac{w_i}{\left(\sum_{i=1}^n w_i^3 \right)^{\frac{1}{3}}}$.*

Corollary 3 (equivalent task for energy) *Consider a fork or join graph G and a deadline D, and assume that the speeds in the optimal solution to* MinEnergy(G, D) *do not exceed s_{max}. We say that the graph G is* equivalent *to the graph $G^{(eq)}$, consisting of a single task $T_0^{(eq)}$ of weight $w_0^{(eq)} = \left(\sum_{i=1}^n w_i^3 \right)^{\frac{1}{3}} + w_0$, because the minimum energy consumption of both graphs are identical:* $\mathbf{minE}(G, D) = \mathbf{minE}(G^{(eq)}, D)$.

4.2.4. Trees

We extend the results on a fork graph for a tree $G = (V, E)$ with $|V| = n + 1$ tasks. Let T_0 be the root of the tree; it has k children tasks, which are each themselves the root of a tree. A tree can therefore be seen as a fork graph, where the tasks of the fork are trees.

The previous results for fork graphs naturally lead to an algorithm that peels off branches of the tree, starting with the leaves, and replaces each fork subgraph in the tree, composed of a root T_0 and k children, by one task (as in Corollary 3) that becomes the unique child of T_0's parent in the tree. We say that this task is *equivalent* to the fork graph, since the optimal energy consumption will be the same. The computation of the *equivalent* cost of this task is done thanks to a call to the **eq** procedure, while the **tree** procedure computes the solution to MinEnergy(G, D) (see Algorithm 1). Note that the algorithm computes the minimum energy for a tree, but it does not return the speeds at which each task must be executed. However, the algorithm returns the speed of the root task, and it is then straightforward to compute the speed of each children of the root task, and so on.

Theorem 2 (tree graphs) *When G is a tree rooted in T_0 ($T_0 \in V$, where V is the set of tasks), the optimal solution to* MinEnergy(G, D) *can be computed in polynomial time $O(|V|^2)$.*

Proof. Let G be a tree graph rooted in T_0. The optimal solution to MinEnergy(G, D) is obtained with a call to **tree** (G, T_0, D), and we prove its optimality recursively on the depth of the tree. Similarly to the case of the fork graphs, we reduce the tree to an equivalent task that, if executed alone within a deadline D, consumes exactly the same amount of energy. The procedure **eq** is the procedure that reduces a tree to its equivalent task (see Algorithm 1).

Algorithm 1: Solution to MinEnergy(G, D) for trees.

procedure **tree** (tree G, root T_0, deadline D)
begin
  Let w=**eq** (tree G, root T_0);
  if $\frac{w}{D} \leq s_{max}$ **then**
    return $\frac{w^3}{D^2}$;
  else
    if $\frac{w_0}{s_{max}} > D$ **then**
      return *Error:No Solution;*
    else
      /* T_0 *is executed at speed* s_{max} */
      return $w_0 \times s_{max}^2 + \sum_{G_i \text{ subtree rooted in } T_i \in children(T_0)}$ **tree** $\left(G_i, T_i, D - \frac{w_0}{s_{max}}\right)$;
    end
  end
end

procedure **eq** (tree G, root T_0)
begin
  if $children(T_0)=\emptyset$ **then**
    return w_0;
  else
    return $\left(\sum_{G_i \text{ subtree rooted in } T_i \in children(T_0)} (\mathbf{eq}(G_i, T_i))^3\right)^{\frac{1}{3}} + w_0$;
  end
end

If the tree has depth 0, then it is a single task, **eq** (G, T_0) returns the equivalent cost w_0, and the optimal execution speed is $\frac{w_0}{D}$ (see Proposition 1). There is a solution if and only if this speed is not greater than s_{max}, and then the corresponding energy consumption is $\frac{w_0^3}{D^2}$, as returned by the algorithm.

Assume now that for any tree of depth $i < p$, **eq** computes its equivalent cost, and **tree** returns its optimal energy consumption. We consider a tree G of depth p rooted in T_0: $G = T_0 \cup \{G_i\}$, where each subgraph G_i is a tree, rooted in T_i, of maximum depth $p - 1$. As in the case of forks, we know that each subtree G_i has a deadline $D - x$, where $x = \frac{w_0}{s_0}$, and s_0 is the speed at which task T_0 is executed. By induction hypothesis, we suppose that each graph G_i is equivalent to a single task, T_i', of cost w_i' (as computed by the procedure **eq**). We can then use the results obtained on forks to compute $w_0^{(eq)}$ (see proof of Theorem 1):

$$w_0^{(eq)} = \left(\sum_i (w_i')^3 \right)^{\frac{1}{3}} + w_0.$$

Finally the tree is equivalent to one task of cost $w_0^{(eq)}$, and if $\frac{w_0^{(eq)}}{D} \leq s_{max}$, the energy consumption is $\frac{\left(w_0^{(eq)}\right)^3}{D^2}$, and no speed exceeds s_{max}.

Note that the speed of a task is always greater than the speed of its successors. Therefore, if $\frac{w_0^{(eq)}}{D} > s_{max}$, we execute the root of the tree at speed s_{max} and then process each subtree G_i independently. Of course, there is no solution if $\frac{w_0}{s_{max}} > D$, and otherwise we perform the recursive calls to **tree** to process each subtree independently. Their deadline is then $D - \frac{w_0}{s_{max}}$.

To study the time complexity of this algorithm, first note that when calling **tree** (G, T_0, D), there might be at most $|V|$ recursive calls to **tree**, once at each node of the tree. Without accounting for the recursive calls, the **tree** procedure performs one call to the **eq** procedure, which computes the cost of the equivalent task. This **eq** procedure takes a time $O(|V|)$, since we have to consider the $|V|$ tasks, and we add the costs one by one. Therefore, the overall complexity is in $O(|V|^2)$. □

4.2.5. Series-parallel graphs

We can further generalize our results to series-parallel graphs (SPGs), which are built from a sequence of compositions (parallel or series) of smaller-size SPGs. The smallest SPG consists of two nodes connected by an edge (such a graph is called an *elementary SPG*). The first node is the source, while the second one is the sink of the SPG. When composing two SGPs in series, we merge the sink of the first SPG with the source of the second one. For a parallel composition, the two sources are merged, as well as the two sinks, as illustrated in Figure 2.

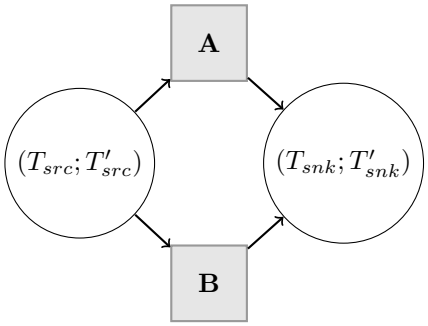
We can extend the results for tree graphs to SPGs, by replacing step by step the SPGs by an equivalent task (procedure **cost** in Algorithm 2): we can compute the equivalent cost for a series or parallel composition.

However, since it is no longer true that the speed of a task is always larger than the speed of its successor (as was the case in a tree), we have not been able to find a recursive property on the tasks that should be set to s_{max}, when one of the speeds obtained with the previous method exceeds s_{max}. The problem of computing a closed form for a SPG with a finite value of s_{max} remains open. Still, we have the following result when $s_{max} = +\infty$:

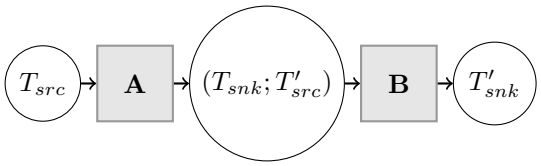
Theorem 3 (series-parallel graphs) *When G is a SPG, it is possible to compute recursively a closed form expression of the optimal solution of* MinEnergy(G, D)*, assuming $s_{max} = +\infty$, in polynomial time $O(|V|)$, where V is the set of tasks.*

Proof. Let G be a series-parallel graph. The optimal solution to MinEnergy(G, D) is obtained with a call to **SPG** (G, D), and we prove its optimality recursively. Similarly to trees, the main idea is to peel the graph off, and to transform it until there remains only a single equivalent task that, if executed alone within a deadline D, would consume exactly

(a) Two SPGs before composition.



(b) Parallel composition.



(c) Series composition.

Figure 2: Composition of series-parallel graphs (SPGs).

the same amount of energy. The procedure **cost** is the procedure that reduces a tree to its equivalent task (see Algorithm 2).

The proof is done by induction on the number of compositions required to build the graph G, p. If $p = 0$, G is an elementary SPG consisting in two tasks, the source T_0 and the sink T_1. It is therefore a linear chain, and therefore equivalent to a single task whose cost is the sum of both costs, w_0+w_1 (see Corollary 1 for linear chains). The procedure **cost** returns therefore the correct equivalent cost, and **SPG** returns the minimum energy consumption.

Let us assume that the procedures return the correct equivalent cost and minimum energy consumption for any SPG consisting of $i < p$ compositions. We consider a SPG G, with p compositions. By definition, G is a composition of two smaller-size SPGs, G_1 and G_2, and both of these SPGs have strictly fewer than p compositions. We consider G'_1 and G'_2, which are identical to G_1 and G_2, except that the cost of their source and sink tasks are set to 0 (these costs are handled separately), and we can reduce both of these SPGs to an equivalent task, of respective costs w'_1 and w'_2, by induction hypothesis. There are two cases:

- If G is a series composition, then after the reduction of G'_1 and G'_2, we have a linear chain in which we consider the source T_0 of G_1, the sink T_1 of G_1 (which is also the source of G_2), and the sink T_2 of G_2. The equivalent cost is therefore $w_0 + w'_1 + w_1 + w'_2 + w_2$, thanks to Corollary 1 for linear chains.
- If G is a parallel composition, the resulting graph is a fork-join graph, and we can use Corollaries 1 and 3 to compute the cost of the equivalent task, accounting for the source T_0 and the sink T_1: $w_0 + \left((w'_1)^3 + (w'_2)^3\right)^{\frac{1}{3}} + w_1$.

Algorithm 2: Solution to MinEnergy(G, D) for series-parallel graphs.

```
procedure SPG (series-parallel graph G, deadline D)
begin
    return (cost(G))^3 / D^2;
end

procedure cost (series-parallel graph G)
begin
    Let T_0 be the source of G and T_1 its sink;
    if G is composed of only two tasks, T_0 and T_1 then
        return w_0 + w_1;
    else
        /* G is a composition of two SPGs G_1 and G_2. */
        For i = 1, 2, let G'_i = G_i where the cost of source and sink tasks is set to 0;
        w'_1 = cost(G'_1); w'_2 = cost(G'_2);
        if G is a series composition then
            Let T_0 be the source of G_1, T_1 be its sink, and T_2 be the sink of G_2;
            return w_0 + w'_1 + w_1 + w'_2 + w_2;
        else
            /* It is a parallel composition. */
            Let T_0 be the source of G, and T_1 be its sink;
            return w_0 + ((w'_1)^3 + (w'_2)^3)^(1/3) + w_1;
        end
    end
end
```

Once the cost of the equivalent task of the SPG has been computed with the call to **cost** (G), the optimal energy consumption is $\frac{(\mathbf{cost}(G))^3}{D^2}$.

Contrarily to the case of tree graphs, since we never need to call the **SPG** procedure again because there is no constraint on s_{max}, the time complexity of the algorithm is the complexity of the **cost** procedure. There is exactly one call to **cost** for each composition, and the number of compositions in the SPG is in $O(|V|)$. All operations in **cost** can be done in $O(1)$, hence a complexity in $O(|V|)$. □

4.3. General DAGs

For arbitrary execution graphs, we can rewrite the MinEnergy(G, D) problem as follows:

$$\begin{array}{lll} \text{Minimize} & & \sum_{i=1}^{n} u_i^{-2} \times w_i \\ \text{subject to} & \text{(i)} & b_i + w_i \times u_i \leq b_j \text{ for each edge } (T_i, T_j) \in E \\ & \text{(ii)} & b_i + w_i \times u_i \leq D \text{ for each task } T_i \in V \\ & \text{(iii)} & u_i \geq \frac{1}{s_{max}} \text{ for each task } T_i \in V \\ & \text{(iv)} & b_i \geq 0 \text{ for each task } T_i \in V \end{array} \tag{2}$$

Here, $u_i = 1/s_i$ is the inverse of the speed to execute task T_i. We now have a convex optimization problem to solve, with linear constraints in the non-negative variables u_i and b_i. In fact, the objective function is a posynomial, so we have a geometric programming problem [16, Section 4.5] for which efficient numerical schemes exist. In addition, such an optimization problem with a smooth convex objective function is known to be well-conditioned [35].

However, as illustrated on simple fork graphs, the optimal speeds are not expected to be rational numbers but instead arbitrarily complex expressions (we have the cubic root of the sum of cubes for forks, and nested expressions of this form for trees). From a computational complexity point of view, we do not know how to encode such numbers in polynomial size of the input (the rational task weights and the execution deadline). Still, we can always solve the problem numerically and get fixed-size numbers that are good approximations of the optimal values.

In the following, we show that the total power consumption of any optimal schedule is constant throughout execution. While this important property does not help to design an optimal solution, it shows that a schedule with large variations in its power consumption is likely to waste a lot of energy.

We need a few notations before stating the result. Consider a schedule for a graph $G = (V, E)$ with n tasks. Task T_i is executed at constant speed s_i (see Lemma 1) and during interval $[b_i, c_i]$: T_i begins its execution at time b_i and completes it at time c_i. The total power consumption $P(t)$ of the schedule at time t is defined as the sum of the power consumed by all tasks executing at time t:

$$P(t) = \sum_{1 \leq i \leq n,\ t \in [b_i, c_i]} s_i^3 \ .$$

Theorem 4. *Consider an instance of* Continuous*, and an optimal schedule for this instance, such that no speed is equal to s_{max}. Then the total power consumption of the schedule throughout execution is constant.*

Proof. We prove this theorem by induction on the number of tasks of the graph. First we prove a preliminary result:

Lemma 2. *Consider a graph $G = (V, E)$ with $n \geq 2$ tasks, and any optimal schedule of deadline D. Let t_1 be the earliest completion time of a task in the schedule. Similarly, let t_2 be the latest starting time of a task in the schedule. Then, either G is composed of independent tasks, or $0 < t_1 \leq t_2 < D$.*

Proof. Task T_i is executed at speed s_i and during interval $[b_i, c_i]$. We have $t_1 = \min_{1 \leq i \leq n} c_i$ and $t_2 = \max_{1 \leq i \leq n} b_i$. Clearly, $0 \leq t_1, t_2 \leq D$ by definition of the schedule. Suppose that $t_2 < t_1$. Let T_1 be a task that ends at time t_1, and T_2 one that starts at time t_2. Then:

- $\nexists T \in V,\ (T_1, T) \in E$ (otherwise, T would start after t_2), therefore, $t_1 = D$;
- $\nexists T \in V,\ (T, T_2) \in E$ (otherwise, T would finish before t_1); therefore $t_2 = 0$.

This also means that all tasks start at time 0 and end at time D. Therefore, G is only composed of independent tasks. □

Back to the proof of the theorem, we consider first the case of a graph with only one task. In an optimal schedule, the task is executed in time D, and at constant speed (Lemma 1), hence with constant power consumption.

Suppose now that the property is true for all DAGs with at most $n-1$ tasks. Let G be a DAG with n tasks. If G is exactly composed of n independent tasks, then we know that the power consumption of G is constant (because all task speeds are constant). Otherwise, let t_1 be the earliest completion time, and t_2 the latest starting time of a task in the optimal schedule. Thanks to Lemma 2, we have $0 < t_1 \leq t_2 < D$.

Suppose first that $t_1 = t_2 = t_0$. There are three kinds of tasks: those beginning at time 0 and ending at time t_0 (set S_1), those beginning at time t_0 and ending at time D (set S_2), and finally those beginning at time 0 and ending at time D (set S_3). Tasks in S_3 execute during the whole schedule duration, at constant speed, hence their contribution to the total power consumption $P(t)$ is the same at each time-step t. Therefore, we can suppress them from the schedule without loss of generality. Next we determine the value of t_0. Let $A_1 = \sum_{T_i \in S_1} w_i^3$, and $A_2 = \sum_{T_i \in S_2} w_i^3$. The energy consumption between 0 and t_0 is $\frac{A_1}{t_0^2}$, and between t_0 and D, it is $\frac{A_2}{(D-t_0)^2}$. The optimal energy consumption is obtained with $t_0 = \frac{A_1^{\frac{1}{3}}}{A_1^{\frac{1}{3}} + A_2^{\frac{1}{3}}}$. Then, the total power consumption of the optimal schedule is the same in both intervals, hence at each time-step: we derive that $P(t) = \left(\frac{A_1^{\frac{1}{3}} + A_2^{\frac{1}{3}}}{D} \right)^3$, which is constant.

Suppose now that $t_1 < t_2$. For each task T_i, let w_i' be the number of operations executed before t_1, and w_i'' the number of operations executed after t_1 (with $w_i' + w_i'' = w_i$). Let G' be the DAG G with execution costs w_i', and G'' be the DAG G with execution costs w_i''. The tasks with a cost equal to 0 are removed from the DAGs. Then, both G' and G'' have strictly fewer than n tasks. We can therefore apply the induction hypothesis. We derive that the power consumption in both DAGs is constant. Since we did not change the speeds of the tasks, the total power consumption $P(t)$ in G is the same as in G' if $t < t_1$, hence a constant. Similarly, the total power consumption $P(t)$ in G is the same as in G'' if $t > t_1$, hence a constant. Considering the same partitioning with t_2 instead of t_1, we show that the total power consumption $P(t)$ is a constant before t_2, and also a constant after t_2. But $t_1 < t_2$, and the intervals $[0, t_2]$ and $[t_1, D]$ overlap. Altogether, the total power consumption is the same constant throughout $[0, D]$, which concludes the proof. □

5. Discrete models

In this section, we present complexity results on the three energy models with a finite number of possible speeds. The only polynomial instance is for the VDD-HOPPING model, for which we write a linear program in Section 5.1. Then, we give NP-completeness results in Section 5.2, and approximation results in Section 5.3, for the DISCRETE and INCREMENTAL models.

5.1. The Vdd-Hopping model

Theorem 5. *With the* VDD-HOPPING *model,* MINENERGY(G, D) *can be solved in polynomial time.*

Proof. Let G be the execution graph of an application with n tasks, and D a deadline. Let $s_1, ..., s_m$ be the set of possible processor speeds. We use the following rational variables: for $1 \leq i \leq n$ and $1 \leq j \leq m$, b_i is the starting time of the execution of task T_i, and $\alpha_{(i,j)}$ is the time spent at speed s_j for executing task T_i. There are $n + n \times m = n(m+1)$ such variables. Note that the total execution time of task T_i is $\sum_{j=1}^{m} \alpha_{(i,j)}$. The constraints are:

- $\forall 1 \leq i \leq n,\ b_i \geq 0$: starting times of all tasks are non-negative numbers;
- $\forall 1 \leq i \leq n,\ b_i + \sum_{j=1}^{m} \alpha_{(i,j)} \leq D$: the deadline is not exceeded by any task;
- $\forall 1 \leq i, i' \leq n$ such that $T_i \to T_{i'},\ b_i + \sum_{j=1}^{m} \alpha_{(i,j)} \leq b_{i'}$: a task cannot start before its predecessor has completed its execution;
- $\forall 1 \leq i \leq n,\ \sum_{j=1}^{m} \alpha_{(i,j)} \times s_j \geq w_i$: task T_i is completely executed.

The objective function is then min $\left(\sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{(i,j)} s_j^3\right)$.

The size of this linear program is clearly polynomial in the size of the instance, all $n(m+1)$ variables are rational, and therefore it can be solved in polynomial time [36]. □

5.2. NP-completeness results

Theorem 6. *With the* INCREMENTAL *model (and hence the* DISCRETE *model),* MINENERGY(G, D) *is NP-complete.*

Proof. We consider the associated decision problem: given an execution graph, a deadline, and a bound on the energy consumption, can we find an execution speed for each task such that the deadline and the bound on energy are respected? The problem is clearly in NP: given the execution speed of each task, computing the execution time and the energy consumption can be done in polynomial time.

To establish the completeness, we use a reduction from 2-Partition [37]. We consider an instance $\mathcal{I}_1$ of 2-Partition: given n strictly positive integers $a_1, \ldots, a_n$, does there exist a subset I of $\{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? Let $T = \frac{1}{2} \sum_{i=1}^{n} a_i$.

We build the following instance $\mathcal{I}_2$ of our problem: the execution graph is a linear chain with n tasks, where:

- task T_i has size $w_i = a_i$;
- the processor can run at $m = 2$ different speeds;

- $s_1 = 1$ and $s_2 = 2$, (i.e., $s_{min} = 1, s_{max} = 2, \delta = 1$);
- $L = 3T/2$;
- $E = 5T$.

Clearly, the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$.

Suppose first that instance $\mathcal{I}_1$ has a solution I. For all $i \in I$, T_i is executed at speed 1, otherwise it is executed at speed 2. The execution time is then $\sum_{i\in I} a_i + \sum_{i\notin I} a_i/2 = \frac{3}{2}T = D$, and the energy consumption is $E = \sum_{i\in I} a_i + \sum_{i\notin I} a_i \times 2^2 = 5T = E$. Both bounds are respected, and therefore the execution speeds are a solution to $\mathcal{I}_2$.

Suppose now that $\mathcal{I}_2$ has a solution. Since we consider the DISCRETE and INCREMENTAL models, each task run either at speed 1, or at speed 2. Let $I = \{i \mid T_i \text{ is executed at speed } 1\}$. Note that we have $\sum_{i\notin I} a_i = 2T - \sum_{i\in I} a_i$.

The execution time is $D' = \sum_{i\in I} a_i + \sum_{i\notin I} a_i/2 = T + (\sum_{i\in I} a_i)/2$. Since the deadline is not exceeded, $D' \leq D = 3T/2$, and therefore $\sum_{i\in I} a_i \leq T$.

For the energy consumption of the solution of $\mathcal{I}_2$, we have $E' = \sum_{i\in I} a_i + \sum_{i\notin I} a_i \times 2^2 = 2T + 3\sum_{i\notin I} a_i$. Since $E' \leq E = 5T$, we obtain $3\sum_{i\notin I} a_i \leq 3T$, and hence $\sum_{i\notin I} a_i \leq T$.

Since $\sum_{i\in I} a_i + \sum_{i\notin I} a_i = 2T$, we conclude that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i = T$, and therefore $\mathcal{I}_1$ has a solution. This concludes the proof. □

5.3. Approximation results

Here we explain, for the INCREMENTAL and DISCRETE models, how the solution to the NP-hard problem can be approximated. Note that, given an execution graph and a deadline, the optimal energy consumption with the CONTINUOUS model is always lower than that with the other models, which are more constrained.

Theorem 7. *With the* INCREMENTAL *model, for any integer $K > 0$, the* MINENERGY(G, D) *problem can be approximated within a factor $(1 + \frac{\delta}{s_{min}})^2(1 + \frac{1}{K})^2$, in a time polynomial in the size of the instance and in K.*

Proof. Consider an instance $\mathcal{I}_{inc}$ of the problem with the INCREMENTAL model. The execution graph G has n tasks, D is the deadline, δ is the minimum permissible speed increment, and s_{min}, s_{max} are the speed bounds. Moreover, let $K > 0$ be an integer, and let E_{inc} be the optimal value of the energy consumption for this instance $\mathcal{I}_{inc}$.

We construct the following instance $\mathcal{I}_{vdd}$ with the VDD-HOPPING model: the execution graph and the deadline are the same as in instance $\mathcal{I}_{inc}$, and the speeds can take the values

$$\left\{ s_{min} \times \left(1 + \frac{1}{K}\right)^i \right\}_{0\leq i\leq N},$$

where N is such that s_{max} is not exceeded: $N = \left\lfloor (\ln(s_{max}) - \ln(s_{min}))/\ln\left(1 + \frac{1}{K}\right) \right\rfloor$. As N is asymptotically of order $O(K \ln(s_{max}))$, the number of possible speeds in $\mathcal{I}_{vdd}$, and hence the size of $\mathcal{I}_{vdd}$, is polynomial in the size of $\mathcal{I}_{inc}$ and K.

Next, we solve $\mathcal{I}_{vdd}$ in polynomial time thanks to Theorem 5. For each task T_i, let $s_i^{(vdd)}$ be the average speed of T_i in this solution: if the execution time of the task in the solution

is d_i, then $s_i^{(vdd)} = w_i/d_i$; E_{vdd} is the optimal energy consumption obtained with these speeds. Let $s_i^{(algo)} = \min_u\{s_{min} + u \times \delta \mid s_{min} + u \times \delta \geq s_i^{(vdd)}\}$ be the smallest speed in $\mathcal{I}_{inc}$ that is larger than $s_i^{(vdd)}$. There exists such a speed since, because of the values chosen for $\mathcal{I}_{vdd}$, $s_i^{(vdd)} \leq s_{max}$. The values $s_i^{(algo)}$ can be computed in time polynomial in the size of $\mathcal{I}_{inc}$ and K. Let E_{algo} be the energy consumption obtained with these values.

In order to prove that this algorithm is an approximation of the optimal solution, we need to prove that $E_{algo} \leq (1+\frac{\delta}{s_{min}})^2(1+\frac{1}{K})^2 \times E_{inc}$. For each task T_i, $s_i^{(algo)} - \delta \leq s_i^{(vdd)} \leq s_i^{(algo)}$. Since $s_{min} \leq s_i^{(vdd)}$, we derive that $s_i^{(algo)} \leq s_i^{(vdd)} \times (1+\frac{\delta}{s_{min}})$. Summing over all tasks, we get

$$E_{algo} = \sum_i w_i \left(s_i^{(algo)}\right)^2 \leq \sum_i w_i \left(s_i^{(vdd)} \times (1+\tfrac{\delta}{s_{min}})\right)^2 \leq E_{vdd} \times \left(1+\tfrac{\delta}{s_{min}}\right)^2.$$

Next, we bound E_{vdd} thanks to the optimal solution with the CONTINUOUS model, E_{con}. Let $\mathcal{I}_{con}$ be the instance where the execution graph G, the deadline D, the speeds s_{min} and s_{max} are the same as in instance $\mathcal{I}_{inc}$, but now admissible speeds take any value between s_{min} and s_{max}. Let $s_i^{(con)}$ be the optimal continuous speed for task T_i, and let $0 \leq u \leq N$ be the value such that:

$$s_{min} \times \left(1+\tfrac{1}{K}\right)^u \leq s_i^{(con)} \leq s_{min} \times \left(1+\tfrac{1}{K}\right)^{u+1} = s_i^* \ .$$

In order to bound the energy consumption for I_{vdd}, we assume that T_i runs at speed s_i^*, instead of $s_i^{(vdd)}$. The solution with these speeds is a solution to I_{vdd}, and its energy consumption is $E^* \geq E_{vdd}$. From the previous inequalities, we deduce that $s_i^* \leq s_i^{(con)} \times \left(1+\frac{1}{K}\right)$, and by summing over all tasks,

$$\begin{aligned} E_{vdd} \leq E^* &= \sum_i w_i \left(s_i^*\right)^2 \leq \sum_i w_i \left(s_i^{(con)} \times \left(1+\tfrac{1}{K}\right)\right)^2 \\ &\leq E_{con} \times \left(1+\tfrac{1}{K}\right)^2 \leq E_{inc} \times \left(1+\tfrac{1}{K}\right)^2 \ . \end{aligned}$$

□

Proposition 3.

- *For any integer $\delta > 0$, any instance of* MINENERGY(G, D) *with the* CONTINUOUS *model can be approximated within a factor $(1+\frac{\delta}{s_{min}})^2$ in the* INCREMENTAL *model with speed increment δ.*
- *For any integer $K > 0$, any instance of* MINENERGY(G, D) *with the* DISCRETE *model can be approximated within a factor $(1+\frac{\alpha}{s_1})^2(1+\frac{1}{K})^2$, with $\alpha = \max_{1\leq i<m}\{s_{i+1} - s_i\}$, in a time polynomial in the size of the instance and in K.*

Proof. For the first part, let $s_i^{(con)}$ be the optimal continuous speed for task T_i in instance $\mathcal{I}_{con}$; E_{con} is the optimal energy consumption. For any task T_i, let s_i be the speed of $\mathcal{I}_{inc}$ such that $s_i - \delta < s_i^{con} \leq s_i$. Then, $s_i^{(con)} \leq s_i \times \left(1+\frac{\delta}{s_{min}}\right)$. Let E be the energy with speeds s_i. $E_{con} \leq E \times \left(1+\frac{\delta}{s_{min}}\right)^2$. Let E_{inc} be the optimal energy of $\mathcal{I}_{inc}$. Then, $E_{con} \leq E_{inc} \times \left(1+\frac{\delta}{s_{min}}\right)^2$.

For the second part, we use the same algorithm as in Theorem 7. The same proof leads to the approximation ratio with α instead of δ. □

6. Conclusion

In this paper, we have assessed the tractability of a classical scheduling problem, with task preallocation, under various energy models. We have given several results related to CONTINUOUS speeds. However, while these are of conceptual importance, they cannot be achieved with physical devices, and we have analyzed several models enforcing a bounded number of achievable speeds, a.k.a. modes. In the classical DISCRETE model that arises from DVFS techniques, admissible speeds can be irregularly distributed, which motivates the VDD-HOPPING approach that mixes two consecutive modes optimally. While computing optimal speeds is NP-hard with discrete modes, it has polynomial complexity when mixing speeds. Intuitively, the VDD-HOPPING approach allows for smoothing out the discrete nature of the modes. An alternate (and simpler in practice) solution to VDD-HOPPING is the INCREMENTAL model, where one sticks with unique speeds during task execution as in the DISCRETE model, but where consecutive modes are regularly spaced. Such a model can be made arbitrarily efficient, according to our approximation results.

Altogether, this paper has laid the theoretical foundations for a comparative study of energy models. In the recent years, we have observed an increased concern for green computing, and a rapidly growing number of approaches. It will be very interesting to see which energy-saving technological solutions will be implemented in forthcoming future processor chips.

Regardless of the (future) energy model, there are two important future research directions that can already be envisioned:

- For those situations where the optimal solutions or approximation algorithms provided in this paper would be too costly, fast heuristics can easily be introduced. Typically, such heuristics would greedily perform local changes in the schedule until a local optimum has been reached. It would be very interesting to assess the energy savings achieved by such "fast" solutions with respect to the gain provided by the optimal solution.

- This paper has dealt with a fixed (given) mapping of the task graph. In some situations, the user may well have the possibility to choose, say, the list-schedule that assigns tasks to physical resources. Given a deadline, the problem is already NP-complete without energy considerations. Introducing variable speeds together with an energy-oriented objective dramatically increases the combinatorial difficulty of the problem. Still, designing and evaluating fast yet efficient heuristics would be of great practical significance.

Acknowledgement. We thank the reviewers for their observations and suggestions that greatly improved the final version of the paper.

REFERENCES

1. Mills MP. The internet begins with coal. *Environment and Climate News* 1999; :.
2. Ge R, Feng X, Cameron KW. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. *Proceedings of the ACM/IEEE conference on SuperComputing (SC)*, IEEE Computer Society, 2005; 34.
3. Skadron K, Stan MR, Sankaranarayanan K, Huang W, Velusamy S, Tarjan D. Temperature-aware microarchitecture: modeling and implementation. *ACM Transactions on Architecture and Code Optimization* 2004; **1**(1):94–125.
4. Hotta Y, Sato M, Kimura H, Matsuoka S, Boku T, Takahashi D. Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE Computer Society Press: Los Alamitos, CA, USA, 2006; 340, doi:http://doi.ieeecomputersociety.org/10.1109/IPDPS.2006.1639597.
5. Ishihara T, Yasuura H. Voltage scheduling problem for dynamically variable voltage processors. *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, ACM Press, 1998; 197–202.
6. Pruhs K, van Stee R, Uthaisombut P. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems* 2008; **43**:67–80.
7. Chandrakasan AP, Sinha A. JouleTrack: A Web Based Tool for Software Energy Profiling. *Design Automation Conference*, IEEE Computer Society Press: Los Alamitos, CA, USA, 2001; 220–225.
8. Aydin H, Yang Q. Energy-aware partitioning for multiprocessor real-time systems. *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE CS Press, 2003; 113–121.
9. Chen JJ, Kuo TW. Multiprocessor energy-efficient scheduling for real-time tasks. *Proceedings of International Conference on Parallel Processing (ICPP)*, IEEE CS Press, 2005; 13–20.
10. Rayward-Smith VJ, Burton FW, Janacek GJ. Scheduling parallel programs assuming preallocation. *Scheduling Theory and its Applications*, Chrétienne P, Coffman Jr EG, Lenstra JK, Liu Z (eds.), John Wiley and Sons, 1995.
11. Wang L, von Laszewski G, Dayal J, Wang F. Towards Energy Aware Scheduling for Precedence Constrained Parallel Tasks in a Cluster with DVFS. *Proceedings of CCGrid'2010, the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010; 368 –377, doi:10.1109/CCGRID.2010.19.
12. Prathipati RB. Energy efficient scheduling techniques for real-time embedded systems. Master's Thesis, Texas A&M University May 2004.
13. Bansal N, Kimbrel T, Pruhs K. Speed scaling to manage energy and temperature. *Journal of the ACM* 2007; **54**(1):1 – 39, doi:http://doi.acm.org/10.1145/1206035.1206038.
14. Okuma T, Yasuura H, Ishihara T. Software energy reduction techniques for variable-voltage processors. *Design Test of Computers, IEEE* Mar 2001; **18**(2):31 –41, doi:10.1109/54.914613.
15. Miermont S, Vivet P, Renaudin M. A Power Supply Selector for Energy- and Area-Efficient Local Dynamic Voltage Scaling. *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, *Lecture Notes in Computer Science*, vol. 4644, Azémard N, Svensson L (eds.). Springer Berlin / Heidelberg, 2007; 556–565. URL `http://dx.doi.org/10.1007/978-3-540-74442-9_54`.
16. Boyd S, Vandenberghe L. *Convex Optimization*. Cambridge University Press, 2004.
17. Lee S, Sakurai T. Run-time voltage hopping for low-power real-time systems. *Proceedings of DAC'2000, the 37th Conference on Design Automation*, 2000; 806–809.
18. Lahiri K, Raghunathan A, Dey S, Panigrahi D. Battery-driven system design: a new frontier in low power design. *Proceedings of ASP-DAC 2002, the 7th Asia and South Pacific Design Automation Conference and the 15th International Conference on VLSI Design*, 2002; 261 –267, doi:10.1109/ASPDAC.2002.994932.
19. Grosse P, Durand Y, Feautrier P. Methods for power optimization in SOC-based data flow systems. *ACM Trans. Des. Autom. Electron. Syst.* June 2009; **14**:38:1–38:20, doi:http://doi.acm.org/10.1145/1529255.1529260. URL `http://doi.acm.org/10.1145/1529255.1529260`.
20. Jejurikar R, Pereira C, Gupta R. Leakage aware dynamic voltage scaling for real-time embedded systems. *Proceedings of DAC'04, the 41st annual Design Automation Conferencea*, ACM: New York, NY, USA, 2004; 275–280, doi:http://doi.acm.org/10.1145/996566.996650.
21. Chen JJ, Kuo CF. Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scaling (DVS) Platforms. *Proceedings of the International Workshop on Real-Time Computing Systems and Applications*, IEEE Computer Society: Los Alamitos, CA, USA, 2007; 28–38, doi:http://doi.ieeecomputersociety.org/10.1109/RTCSA.2007.37.
22. Kim KH, Buyya R, Kim J. Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters. *Proceedings of CCGRID 2007, the 7th IEEE International*

Symposium on Cluster Computing and the Grid, 2007; 541 –548, doi:10.1109/CCGRID.2007.85.
23. Yang L, Man L. On-Line and Off-Line DVS for Fixed Priority with Preemption Threshold Scheduling. *Proceedings of ICESS'09, the International Conference on Embedded Software and Systems*, 2009; 273 –280, doi:10.1109/ICESS.2009.50.
24. AMD processors. http://www.amd.com.
25. Intel XScale technology. http://www.intel.com/design/intelxscale.
26. Langen P, Juurlink B. Leakage-aware multiprocessor scheduling. *J. Signal Process. Syst.* 2009; **57**(1):73–88, doi:http://dx.doi.org/10.1007/s11265-008-0176-8.
27. Chen G, Malkowski K, Kandemir M, Raghavan P. Reducing power with performance constraints for parallel sparse applications. *Proceedings of IPDPS 2005, the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005; 8 pp., doi:10.1109/IPDPS.2005.378.
28. Xu R, Mossé D, Melhem R. Minimizing expected energy consumption in real-time systems through dynamic voltage scaling. *ACM Trans. Comput. Syst.* 2007; **25**(4):9, doi:http://doi.acm.org/10.1145/1314299.1314300.
29. Gonzalez R, Horowitz M. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits* Sep 1996; **31**(9):1277 –1284, doi:10.1109/4.535411.
30. Yao F, Demers A, Shenker S. A scheduling model for reduced CPU energy. *Proceedings of FOCS '95, the 36th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society: Washington, DC, USA, 1995; 374.
31. Zhang Y, Hu XS, Chen DZ. Task scheduling and voltage selection for energy minimization. *Proceedings of DAC'02, the 39th annual Design Automation Conference*, ACM: New York, NY, USA, 2002; 183–188, doi:http://doi.acm.org/10.1145/513918.513966.
32. Beigne E, Clermidy F, Durupt J, Lhermet H, Miermont S, Thonnart Y, Xuan T, Valentian A, Varreau D, Vivet P. An asynchronous power aware and adaptive NoC based circuit. *Proceedings of the 2008 IEEE Symposium on VLSI Circuits*, 2008; 190–191, doi:10.1109/VLSIC.2008.4586002.
33. Beigne E, Clermidy F, Miermont S, Thonnart Y, Valentian A, Vivet P. A Localized Power Control mixing hopping and Super Cut-Off techniques within a GALS NoC. *Proceedings of ICICDT 2008, the IEEE International Conference on Integrated Circuit Design and Technology and Tutorial*, 2008; 37–42, doi:10.1109/ICICDT.2008.4567241.
34. Kawaguchi H, Zhang G, Lee S, Sakurai T. An LSI for VDD-Hopping and MPEG4 System Based on the Chip. *Proceedings of ISCAS'2001, the International Symposium on Circuits and Systems*, 2001.
35. Nocedal J, Wright SJ. *Numerical Optimization*. Springer, 2006.
36. Schrijver A. *Combinatorial Optimization: Polyhedra and Efficiency*, *Algorithms and Combinatorics*, vol. 24. Springer-Verlag, 2003.
37. Garey MR, Johnson DS. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.: New York, NY, USA, 1990.